Computer Vision

(UCS522)

Lab Assignment -3

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted To: Dr. Shailendra Tiwari

Submitted by : Prinzu Choudhury

Roll Number- 101703403

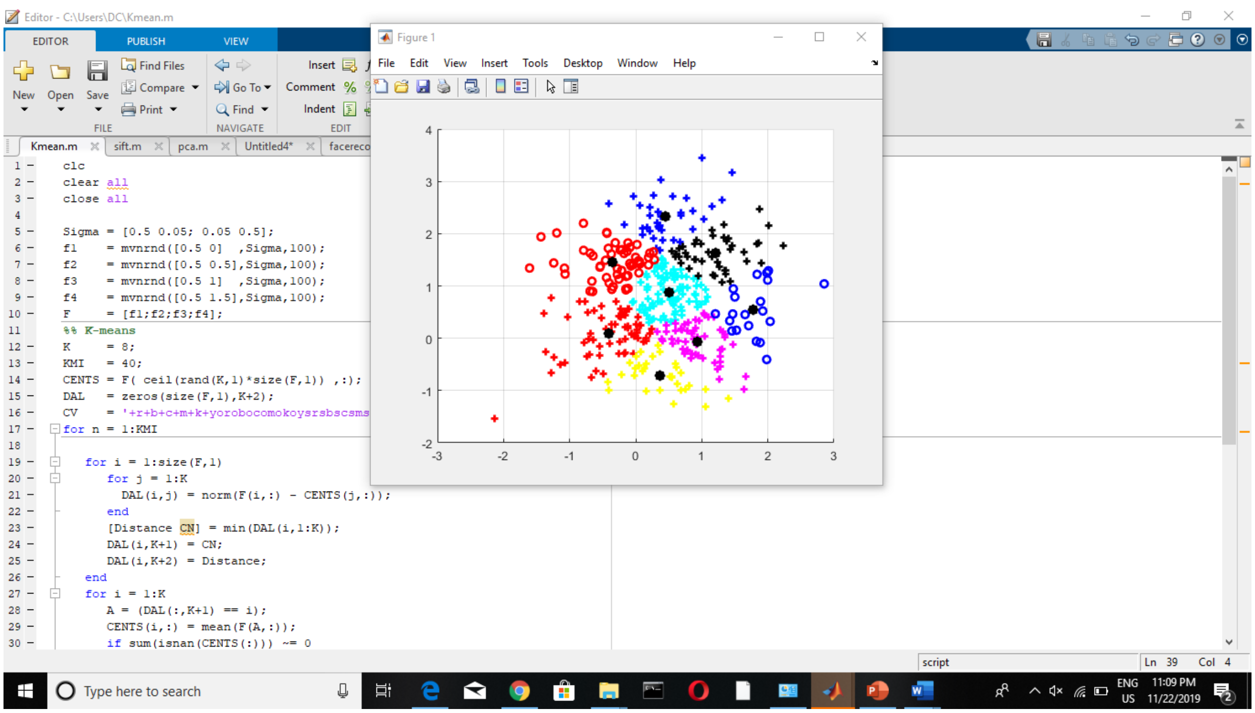
COE18

Q.Implement the following Computer Vision Algorithms using MATLAB/OpenCV/Python/Or any other Platform.

1. Use K-mean/Fuzzy C-mean Clustering techniques in Image segmentation.

K MEANS ALGORITHM

K means algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the inter-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.



CODE-

```
clc
clear all
```

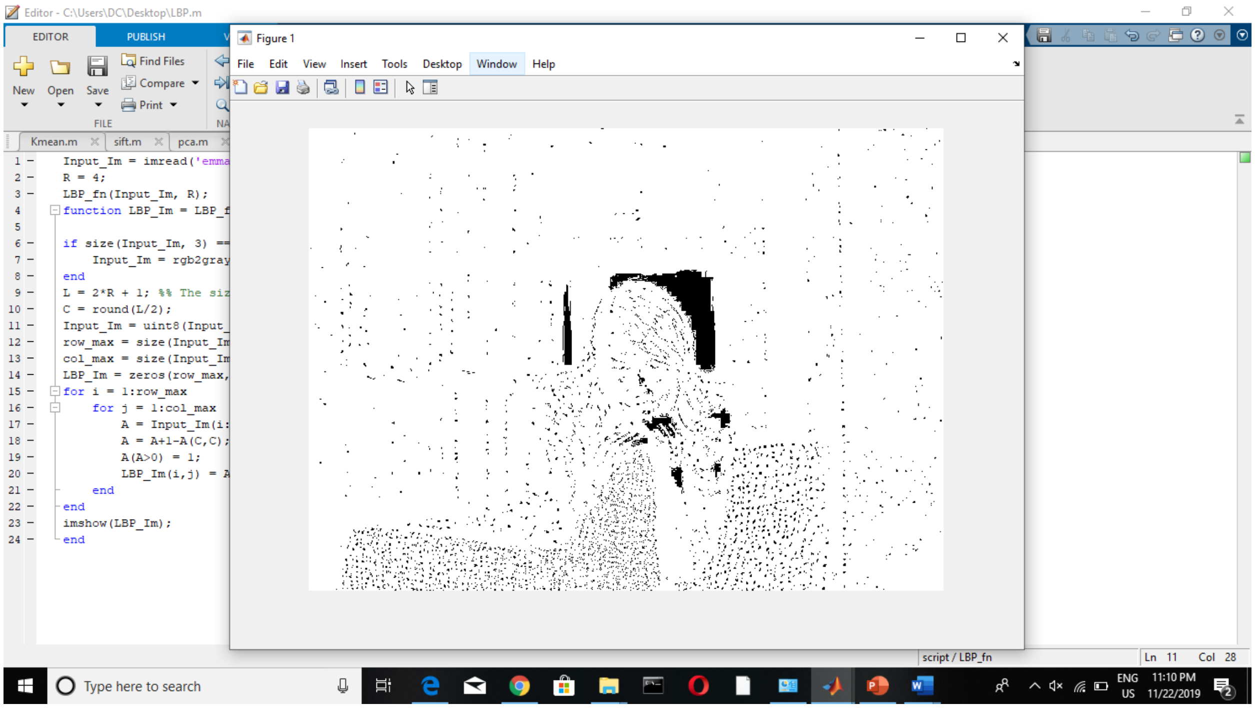
```
close all

Sigma = [0.5 0.05; 0.05 0.5];
f1    = mvnrnd([0.5 0]   ,Sigma,100);
f2    = mvnrnd([0.5 0.5],Sigma,100);
f3    = mvnrnd([0.5 1]   ,Sigma,100);
f4    = mvnrnd([0.5 1.5],Sigma,100);
F     = [f1;f2;f3;f4];

K     = 8;                                                  %
Cluster Numbers
KMI   = 40;                                                 % K-
means Iteration
CENTS = F( ceil(rand(K,1)*size(F,1)) ,:);                   %
Cluster Centers
DAL   = zeros(size(F,1),K+2);                               %
Distances and Labels
CV    = '+r+b+c+m+k+yorobocomokoysrsbscsmsksy';             %
Color Vector

for n = 1:KMI

   for i = 1:size(F,1)
      for j = 1:K
        DAL(i,j) = norm(F(i,:) - CENTS(j,:));
      end
      [Distance CN] = min(DAL(i,1:K));                      % 1:K
are Distance from Cluster Centers 1:K
      DAL(i,K+1) = CN;                                      % K+1
is Cluster Label
      DAL(i,K+2) = Distance;                                % K+2
is Minimum Distance
   end
   for i = 1:K
      A = (DAL(:,K+1) == i);                                %
Cluster K Points
      CENTS(i,:) = mean(F(A,:));                            % New
Cluster Centers
      if sum(isnan(CENTS(:))) ~= 0                          % If
CENTS(i,:) Is Nan Then Replace It With Random Point
         NC = find(isnan(CENTS(:,1)) == 1);                 %
Find Nan Centers
         for Ind = 1:size(NC,1)
         CENTS(NC(Ind),:) = F(randi(size(F,1)),:);
```

```
        end
      end
    end


clf
figure(1)
hold on

 for i = 1:K
PT = F(DAL(:,K+1) == i,:);                            %
Find points of each cluster
plot(PT(:,1),PT(:,2),CV(2*i-1:2*i),'LineWidth',2);    %
Plot points with determined color and shape
plot(CENTS(:,1),CENTS(:,2),'*k','LineWidth',7);       %
Plot cluster centers
 end

hold off
grid on
pause(0.1)

end
```

2. Linear Binary Pattern (LBP)/ Linear Ternary Pattern (LTP)/and its variant.

LBD ALGORITHM

LBG algorithm is like a K-means clustering algorithm which takes a set of input vectors S = {xi ∈ Rd | i = 1, 2, . . . , n} as input and generates a representative subset of vectors C = {cj ∈ Rd | j = 1, 2, . . . , K} with a user specified K << n as output according to the similarity measure. For the application of Vector Quantization (VQ), d = 16, K = 256 or 512 are commonly used.



CODE-
```
Input_Im = imread('C:\Users\Prinzu\Desktop\Practice code\pres.jpg');
R = 4;
LBP_fn(Input_Im, R);
function LBP_Im = LBP_fn(Input_Im, R)

if size(Input_Im, 3) == 3
    Input_Im = rgb2gray(Input_Im);
end
L = 2*R + 1; %% The size of the LBP label
C = round(L/2);
Input_Im = uint8(Input_Im);
row_max = size(Input_Im,1)-L+1;
col_max = size(Input_Im,2)-L+1;
LBP_Im = zeros(row_max, col_max);
for i = 1:row_max
    for j = 1:col_max
        A = Input_Im(i:i+L-1, j:j+L-1);
        A = A+1-A(C,C);
        A(A>0) = 1;
        LBP_Im(i,j) = A(C,L) + A(L,L)*2 + A(L,C)*4 + A(L,1)*8 + A(C,1)*16 + A(1,1)*32 + A(1,C)*64 + A(1,L)*128;
    end
end
```

```
imshow(LBP_Im);
end
```

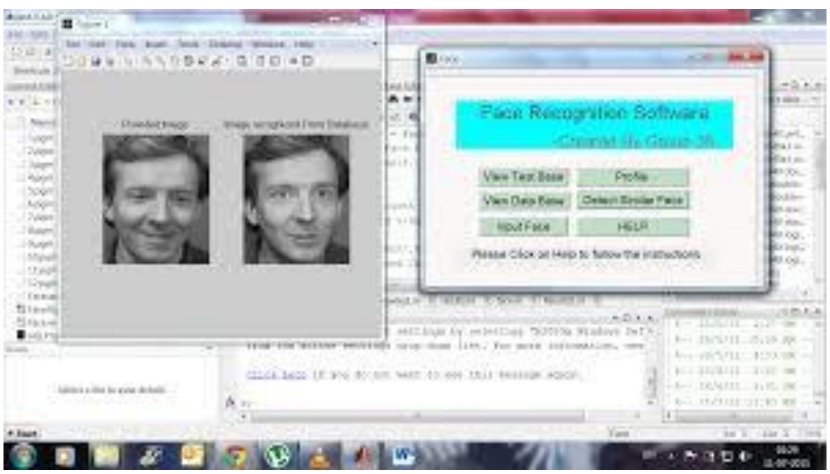
3. Develop an efficient Face recognition system using Principal Component Analysis (PCA)

PCA ALGORITHM

One of the simplest and most effective PCA approaches used in face recognition systems is the so-called eigenface approach. This approach transforms faces into a small set of essential characteristics, eigenfaces, which are the main components of the initial set of learning images (training set). Recognition is done by projecting a new image in the eigenface subspace, after which the person is classified by comparing its position in eigenface space with the position of known individuals [3]. The advantage of this approach over other face recognition systems is in its simplicity, speed and insensitivity to small or gradual changes on the face. The problem is limited to files that can be used to recognize the face. Namely, the images must be vertical frontal views of human faces. The whole recognition process involves two steps:

A. Initialization process

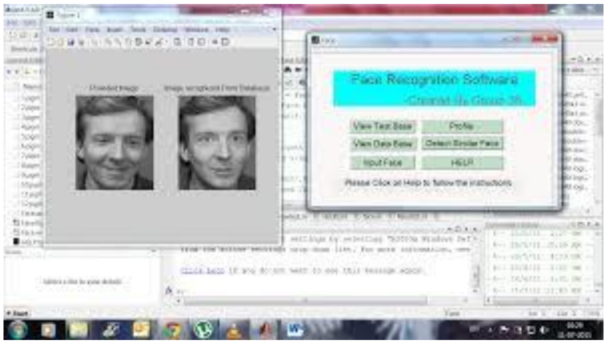
B. Recognition process



Code-

```
clc;
close all;
clear all;
numdata=20; %should be even
%step 1, generating a dataset
x1=rand(numdata/2,1);
y1=rand(numdata/2,1);
x2=3*rand(numdata/2,1)+3;
y2=3*rand(numdata/2,1)+3;
x=[x1;x2];
```

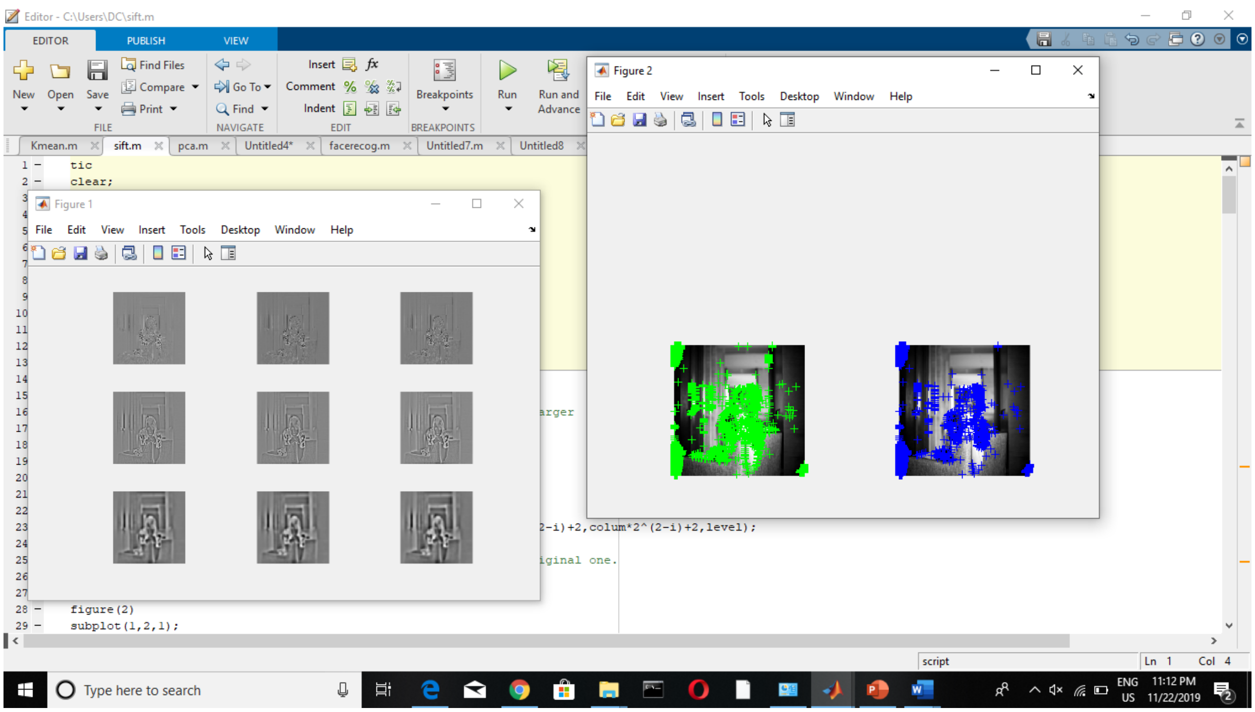
```
y=[y1;y2];
%step 2, finding a mean and subtracting
xmean=mean(x);
ymean=mean(y);
xnew=x-xmean*ones(numdata,1);
ynew=y-ymean*ones(numdata,1);
subplot(3,1,1);
plot(x,y, 'o');
title('Original Data');
%Uncomment to see the data after the deduction of the mean
%subplot(4,1,2);
%plot(xnew,ynew, 'o');
%title('mean is deducted')
%step 3, covariance matrix
covariancematrix=cov(xnew,ynew);
%step 4, Finding Eigenvectors
[V,D] = eig(covariancematrix);
D=diag(D);
maxeigval=V(:,find(D==max(D)));
%step 5, Deriving the new data set
%finding the projection onto the eigenvectors
finaldata=maxeigval'*[xnew,ynew]';
subplot(3,1,2);
stem(finaldata, 'DisplayName', 'finaldata', 'YDataSource',
'finaldata');
title('PCA 1D output ')
%we do a classification now
subplot(3,1,3);
title('Final Classification')
hold on
for i=1:size(finaldata,2)
    if  finaldata(i)>=0
        plot(x(i),y(i),'o')
        plot(x(i),y(i),'r*')

    else
        plot(x(i),y(i),'o')
        plot(x(i),y(i),'g*')
    end

end
```



4. SIFT ALGORITHM

The scale-invariant feature transform (SIFT) is a feature detection algorithm in computer vision to detect and describe local features in images. It was patented in Canada by the University of British Columbia[1] and published by David Lowe in 1999.[2] Applications include object recognition, robotic mapping and navigation, image stitching, 3D modeling, gesture recognition, video tracking, individual identification of wildlife and match moving.

SIFT keypoints of objects are first extracted from a set of reference images[2] and stored in a database. An object is recognized in a new image by individually comparing each feature from the new image to this database and finding candidate matching features based on Euclidean distance of their feature vectors.



CODE-

```
tic
clear;
clc;
row=256;
colum=256;
img=imread('pres.jpg');
img=imresize(img,[row,colum]);
img=rgb2gray(img);
% img=histeq(img);
img=im2double(img);
origin=img;
% img=medfilt2(img);
toc

sigma0=sqrt(2);
octave=3;%6*sigma*k^(octave*level)<=min(m,n)/(2^(octave-2))
level=3;
D=cell(1,octave);
for i=1:octave
D(i)=mat2cell(zeros(row*2^(2-i)+2,colum*2^(2-
i)+2,level),row*2^(2-i)+2,colum*2^(2-i)+2,level);
end

temp_img=kron(img,ones(2));
temp_img=padarray(temp_img,[1,1],'replicate');
figure(2)
subplot(1,2,1);
imshow(origin)
%create the DoG pyramid.
for i=1:octave
    temp_D=D{i};
    for j=1:level
        scale=sigma0*sqrt(2)^(1/level)^((i-1)*level+j);
        p=(level)*(i-1);
        figure(1);
        subplot(octave,level,p+j);
        f=fspecial('gaussian',[1,floor(6*scale)],scale);
        L1=temp_img;
        if(i==1&&j==1)
        L2=conv2(temp_img,f,'same');
        L2=conv2(L2,f','same');
```

```
        temp_D(:,:,j)=L2-L1;
        imshow(uint8(255 * mat2gray(temp_D(:,:,j))));
        L1=L2;
        else
        L2=conv2(temp_img,f,'same');
        L2=conv2(L2,f','same');
        temp_D(:,:,j)=L2-L1;
        L1=L2;
        if(j==level)
            temp_img=L1(2:end-1,2:end-1);
        end
        imshow(uint8(255 * mat2gray(temp_D(:,:,j))));
        end
    end
    D{i}=temp_D;
    temp_img=temp_img(1:2:end,1:2:end);
    temp_img=padarray(temp_img,[1,1],'both','replicate');
end
toc

tic
interval=level-1;
number=0;
for i=2:octave+1
    number=number+(2^(i-octave)*colum)*(2*row)*interval;
end
extrema=zeros(1,4*number);
flag=1;
for i=1:octave
    [m,n,~]=size(D{i});
    m=m-2;
    n=n-2;
    volume=m*n/(4^(i-1));
    for k=2:interval
        for j=1:volume
            % starter=D{i}(x+1,y+1,k);
            x=ceil(j/n);
            y=mod(j-1,m)+1;
            sub=D{i}(x:x+2,y:y+2,k-1:k+1);
            large=max(max(max(sub)));
            little=min(min(min(sub)));
            if(large==D{i}(x+1,y+1,k))
                temp=[i,k,j,1];
                extrema(flag:(flag+3))=temp;
```

```
                flag=flag+4;
            end
            if(little==D{i}(x+1,y+1,k))
                temp=[i,k,j,-1];
                extrema(flag:(flag+3))=temp;
                flag=flag+4;
            end
        end
    end
end
idx= extrema==0;
extrema(idx)=[];
toc
[m,n]=size(img);
x=floor((extrema(3:4:end)-1)./(n./(2.^(extrema(1:4:end)-
2))))+1;
y=mod((extrema(3:4:end)-1),m./(2.^(extrema(1:4:end)-2)))+1;
ry=y./2.^(octave-1-extrema(1:4:end));
rx=x./2.^(octave-1-extrema(1:4:end));
figure(2)
subplot(1,2,2);
imshow(origin)
hold on
plot(ry,rx,'r+');

tic
threshold=0.1;
r=10;
extr_volume=length(extrema)/4;
[m,n]=size(img);
secondorder_x=conv2([-1,1;-1,1],[-1,1;-1,1]);
secondorder_y=conv2([-1,-1;1,1],[-1,-1;1,1]);
for i=1:octave
    for j=1:level
        test=D{i}(:,:,j);
        temp=-
1./conv2(test,secondorder_y,'same').*conv2(test,[-1,-
1;1,1],'same');
        D{i}(:,:,j)=temp.*conv2(test',[-1,-
1;1,1],'same')*0.5+test;
    end
end
for i=1:extr_volume
```

```
    x=floor((extrema(4*(i-1)+3)-1)/(n/(2^(extrema(4*(i-
1)+1)-2))))+1;
    y=mod((extrema(4*(i-1)+3)-1),m/(2^(extrema(4*(i-1)+1)-
2)))+1;
    rx=x+1;
    ry=y+1;
    rz=extrema(4*(i-1)+2);
    z=D{extrema(4*(i-1)+1)}(rx,ry,rz);
    if(abs(z)<threshold)
        extrema(4*(i-1)+4)=0;
    end
end
idx=find(extrema==0);
idx=[idx,idx-1,idx-2,idx-3];
extrema(idx)=[];
extr_volume=length(extrema)/4;
x=floor((extrema(3:4:end)-1)./(n./(2.^(extrema(1:4:end)-
2))))+1;
y=mod((extrema(3:4:end)-1),m./(2.^(extrema(1:4:end)-2)))+1;
ry=y./2.^(octave-1-extrema(1:4:end));
rx=x./2.^(octave-1-extrema(1:4:end));
figure(2)
subplot(2,2,3);
imshow(origin)
hold on
plot(ry,rx,'g+');
for i=1:extr_volume
    x=floor((extrema(4*(i-1)+3)-1)/(n/(2^(extrema(4*(i-
1)+1)-2))))+1;
    y=mod((extrema(4*(i-1)+3)-1),m/(2^(extrema(4*(i-1)+1)-
2)))+1;
    rx=x+1;
    ry=y+1;
    rz=extrema(4*(i-1)+2);
        Dxx=D{extrema(4*(i-1)+1)}(rx-
1,ry,rz)+D{extrema(4*(i-1)+1)}(rx+1,ry,rz)-
2*D{extrema(4*(i-1)+1)}(rx,ry,rz);
        Dyy=D{extrema(4*(i-1)+1)}(rx,ry-
1,rz)+D{extrema(4*(i-1)+1)}(rx,ry+1,rz)-2*D{extrema(4*(i-
1)+1)}(rx,ry,rz);
        Dxy=D{extrema(4*(i-1)+1)}(rx-1,ry-
1,rz)+D{extrema(4*(i-1)+1)}(rx+1,ry+1,rz)-D{extrema(4*(i-
1)+1)}(rx-1,ry+1,rz)-D{extrema(4*(i-1)+1)}(rx+1,ry-1,rz);
        deter=Dxx*Dyy-Dxy*Dxy;
```

```
        R=(Dxx+Dyy)/deter;
        R_threshold=(r+1)^2/r;
        if(deter<0||R>R_threshold)
            extrema(4*(i-1)+4)=0;
        end

end
idx=find(extrema==0);
idx=[idx,idx-1,idx-2,idx-3];
extrema(idx)=[];
extr_volume=length(extrema)/4;
x=floor((extrema(3:4:end)-1)./(n./(2.^(extrema(1:4:end)-
2))))+1;
y=mod((extrema(3:4:end)-1),m./(2.^(extrema(1:4:end)-2)))+1;
ry=y./2.^(octave-1-extrema(1:4:end));
rx=x./2.^(octave-1-extrema(1:4:end));
figure(2)
subplot(2,2,4);
imshow(origin)
hold on
plot(ry,rx,'b+');
toc
%% Orientation Assignment(Multiple orientations assignment)
tic
kpori=zeros(1,36*extr_volume);
minor=zeros(1,36*extr_volume);
f=1;
flag=1;
for i=1:extr_volume
    %search in the certain scale
    scale=sigma0*sqrt(2)^(1/level)^((extrema(4*(i-1)+1)-
1)*level+(extrema(4*(i-1)+2)));
    width=2*round(3*1.5*scale);
    count=1;
    x=floor((extrema(4*(i-1)+3)-1)/(n/(2^(extrema(4*(i-
1)+1)-2))))+1;
    y=mod((extrema(4*(i-1)+3)-1),m/(2^(extrema(4*(i-1)+1)-
2)))+1;
    %make sure the point in the searchable area
    if(x>(width/2)&&y>(width/2)&&x<(m/2^(extrema(4*(i-
1)+1)-2)-width/2-2)&&y<(n/2^(extrema(4*(i-1)+1)-2)-width/2-
2))
        rx=x+1;
        ry=y+1;
```

```
        rz=extrema(4*(i-1)+2);
        reg_volume=width*width;%3? thereom
        % make weight matrix
        weight=fspecial('gaussian',width,1.5*scale);
        %calculate region pixels' magnitude and region
orientation
        reg_mag=zeros(1,count);
        reg_theta=zeros(1,count);
    for l=(rx-width/2):(rx+width/2-1)
        for k=(ry-width/2):(ry+width/2-1)
            reg_mag(count)=sqrt((D{extrema(4*(i-
1)+1)}(l+1,k,rz)-D{extrema(4*(i-1)+1)}(l-
1,k,rz))^2+(D{extrema(4*(i-1)+1)}(l,k+1,rz)-D{extrema(4*(i-
1)+1)}(l,k-1,rz))^2);
            reg_theta(count)=atan2((D{extrema(4*(i-
1)+1)}(l,k+1,rz)-D{extrema(4*(i-1)+1)}(l,k-
1,rz)),(D{extrema(4*(i-1)+1)}(l+1,k,rz)-D{extrema(4*(i-
1)+1)}(l-1,k,rz)))*(180/pi);
            count=count+1;
        end
    end
    %make histogram
    mag_counts=zeros(1,36);
    for x=0:10:359
        mag_count=0;
       for j=1:reg_volume
           c1=-180+x;
           c2=-171+x;
           if(c1<0||c2<0)

if(abs(reg_theta(j))<abs(c1)&&abs(reg_theta(j))>=abs(c2))

mag_count=mag_count+reg_mag(j)*weight(ceil(j/width),mod(j-
1,width)+1);
           end
           else

if(abs(reg_theta(j)>abs(c1)&&abs(reg_theta(j)<=abs(c2))))

mag_count=mag_count+reg_mag(j)*weight(ceil(j/width),mod(j-
1,width)+1);
               end
           end
       end
```

```
            mag_counts(x/10+1)=mag_count;
        end
        % find the max histogram bar and the ones higher than
    80% max
        [maxvm,~]=max(mag_counts);
         kori=find(mag_counts>=(0.8*maxvm));
         kori=(kori*10+(kori-1)*10)./2-180;
         kpori(f:(f+length(kori)-1))=kori;
         f=f+length(kori);
         temp_extrema=[extrema(4*(i-1)+1),extrema(4*(i-
    1)+2),extrema(4*(i-1)+3),extrema(4*(i-1)+4)];

    temp_extrema=padarray(temp_extrema,[0,length(temp_extrema)*
    (length(kori)-1)],'post','circular');
         long=length(temp_extrema);
         minor(flag:flag+long-1)=temp_extrema;
         flag=flag+long;
        end
    end
    idx= minor==0;
    minor(idx)=[];
    extrema=minor;
    % delete unsearchable points and add minor orientation
    points
    idx= kpori==0;
    kpori(idx)=[];
    extr_volume=length(extrema)/4;
    toc
    %% keypoint descriptor
    tic
    d=4;% In David G. Lowe experiment,divide the area into 4*4.
    pixel=4;
    feature=zeros(d*d*8,extr_volume);
    for i=1:extr_volume
        descriptor=zeros(1,d*d*8);% feature dimension is
    128=4*4*8;
        width=d*pixel;
        %x,y centeral point and prepare for location rotation
        x=floor((extrema(4*(i-1)+3)-1)/(n/(2^(extrema(4*(i-
    1)+1)-2))))+1;
        y=mod((extrema(4*(i-1)+3)-1),m/(2^(extrema(4*(i-1)+1)-
    2)))+1;
        z=extrema(4*(i-1)+2);
```

```
        if((m/2^(extrema(4*(i-1)+1)-2)-
pixel*d*sqrt(2)/2)>x&&x>(pixel*d/2*sqrt(2))&&(n/2^(extrema(
4*(i-1)+1)-2)-pixel*d/2*sqrt(2))>y&&y>(pixel*d/2*sqrt(2)))
        sub_x=(x-d*pixel/2+1):(x+d*pixel/2);
        sub_y=(y-d*pixel/2+1):(y+d*pixel/2);
        sub=zeros(2,length(sub_x)*length(sub_y));
        j=1;
        for p=1:length(sub_x)
            for q=1:length(sub_y)
                sub(:,j)=[sub_x(p)-x;sub_y(q)-y];
                j=j+1;
            end
        end
        distort=[cos(pi*kpori(i)/180),-
sin(pi*kpori(i)/180);sin(pi*kpori(i)/180),cos(pi*kpori(i)/1
80)];
    %accordinate after distort
        sub_dis=distort*sub;
        fix_sub=ceil(sub_dis);
        fix_sub=[fix_sub(1,:)+x;fix_sub(2,:)+y];
        patch=zeros(1,width*width);
        for p=1:length(fix_sub)
        patch(p)=D{extrema(4*(i-
1)+1)}(fix_sub(1,p),fix_sub(2,p),z);
        end
        temp_D=(reshape(patch,[width,width]))';
        %create weight matrix.
        mag_sub=temp_D;
        temp_D=padarray(temp_D,[1,1],'replicate','both');
        weight=fspecial('gaussian',width,width/1.5);
        mag_sub=weight.*mag_sub;
        theta_sub=atan((temp_D(2:end-1,3:1:end)-
temp_D(2:end-1,1:1:end-2))./(temp_D(3:1:end,2:1:end-1)-
temp_D(1:1:end-2,2:1:end-1)))*(180/pi);
        % create orientation histogram
        for area=1:d*d
        cover=pixel*pixel;
        ori=zeros(1,cover);
        magcounts=zeros(1,8);
        for angle=0:45:359
          magcount=0;
          for p=1:cover;
              x=(floor((p-1)/pixel)+1)+pixel*floor((area-
1)/d);
```

```
              y=mod(p-1,pixel)+1+pixel*(mod(area-1,d));
              c1=-180+angle;
              c2=-180+45+angle;
              if(c1<0||c2<0)
                  if
(abs(theta_sub(x,y))<abs(c1)&&abs(theta_sub(x,y))>=abs(c2))

                      ori(p)=(c1+c2)/2;
                      magcount=magcount+mag_sub(x,y);
                  end
              else

if(abs(theta_sub(x,y))>abs(c1)&&abs(theta_sub(x,y))<=abs(c2
))
                      ori(p)=(c1+c2)/2;
                      magcount=magcount+mag_sub(x,y);
                  end
              end
          end
          magcounts(angle/45+1)=magcount;
        end
        descriptor((area-1)*8+1:area*8)=magcounts;
        end
        descriptor=normr(descriptor);
        % cap 0.2
        for j=1:numel(descriptor)
            if(abs(descriptor(j))>0.2)
            descriptor(j)=0.2;
            end
        end
        descriptor=normr(descriptor);
        else
            continue;
        end
        feature(:,i)=descriptor';
end
index=find(sum(feature));
feature=feature(:,index);
toc
```

5, Stabilization Write a program to stabilize an input video sequence

One way to stabilize a video is to track a salient feature in the image and use this as an anchor point to cancel out all perturbations relative to it. This procedure, however, must be bootstrapped with knowledge of where such a salient feature lies in the first video frame. In this example, we explore a method of video stabilization that works without any such *a priori* knowledge. It instead automatically searches for the "background plane" in a video sequence, and uses its observed distortion to correct for camera motion.

CODE-

```
filename = 'shaky_car.avi';
hVideoSrc = vision.VideoFileReader(filename, 'ImageColorSpace', 'Intensity');

imgA = step(hVideoSrc); % Read first frame into imgA
imgB = step(hVideoSrc); % Read second frame into imgB

figure; imshowpair(imgA, imgB, 'montage');
title(['Frame A', repmat(' ',[1 70]), 'Frame B']);
```